KAGGLE PROJECT:

Netflix Movies and TV Shows Dataset

By: V ANDAL PRIYADHARSHINI
SRMIST, INDIA

Netflix Movies and TV Shows Dataset

TASK DETAILS :

Recommendation system is required in subscription-based OTT platforms. Recommended engine generally in three types

1.Content Based recommended engine

2.Collaborative recommender engine and

3.Hybrid recommended engine

EXPECTED SUBMISSION:

With the help of this particular data set you have to build a recommended engine.

And your recommended engine will return maximum movies name if an user search for a particular movie.

INTRODUCTION

Recommendation engine with a graph:

The purpose is to build a recommendation engine based on graph by using the Adamic Adar measure.
The more the measure is high, the closest are the two nodes.
The measures between all movies are not pre-calculated, in order to determine the list of recommendation films, we are going to explore the neighborhood of the target film.

Netflix Movies and TV Shows Dataset

TASK DETAILS :

Recommendation system is required in subscription-based OTT platforms. Recommended engine generally in three types

1. Content Based recommended engine
2. Collaborative recommender engine and
3. Hybrid recommended engine

EXPECTED SUBMISSION:

With the help of this particular data set you have to build a recommended engine.

And your recommended engine will return maximum movies name if an user search for a particular movie.

STARTING

KAGGLE PROJECT: Netflix Movies and TV Shows DATASET

#V ANDAL PRIYADHARSHINI P20038

In [1]:
```
#import the basic libraries
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math as math
import time
from IPython.display import Markdown,HTML
import matplotlib.gridspec as gridspec # to do the grid of plots
plt.style.use('seaborn')
plt.rcParams['figure.figsize'] = [14,14]
```

In [2]:
```
'''Plotly visualization .'''
import plotly.offline as py
from plotly.offline import iplot, init_notebook_mode
import plotly.graph_objs as go
py.init_notebook_mode(connected = True) # Required to use plotly offline in jupyter notebook
```

Load the data

In [3]:
```
# Load the data
df = pd.read_csv('netflix_titles.csv')
netdata=df
# convert to datetime
df["date_added"] = pd.to_datetime(df['date_added'])
df['year_added'] = df['date_added'].dt.year
df['year'] = df['date_added'].dt.year
df['month'] = df['date_added'].dt.month
df['day'] = df['date_added'].dt.day
# convert columns "director, listed_in, cast and country" in columns that contain a real list
# the strip function is applied on the elements
```

LOADING THE DATA

jupyter V Andal Priyadharshini p20038 Last Checkpoint: a day ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Markdown

Load the data

In [3]:

```
# load the data
df = pd.read_csv('netflix_titles.csv')
netdata=df
# convert to datetime
df["date_added"] = pd.to_datetime(df['date_added'])
df['year_added'] = df['date_added'].dt.year
df['year'] = df['date_added'].dt.year
df['month'] = df['date_added'].dt.month
df['day'] = df['date_added'].dt.day
# convert columns "director, listed_in, cast and country" in columns that contain a real list
# the strip function is applied on the elements
# if the value is NaN, the new column contains a empty list []
netdata['season_count'] = df.apply(lambda x : x['duration'].split(" ")[0] if "Season" in x['duration'] else "", axis = 1)
netdata['duration'] = df.apply(lambda x : x['duration'].split(" ")[0] if "Season" not in x['duration'] else "", axis = 1)
netdata['duration'] =df.apply(lambda x : '0' if x['duration']=='' else x['duration'],axis=1)
netdata['duration'] =  df['duration'].astype(float)
df['directors'] = df['director'].apply(lambda l: [] if pd.isna(l) else [i.strip() for i in l.split(",")])
df['categories'] = df['listed_in'].apply(lambda l: [] if pd.isna(l) else [i.strip() for i in l.split(",")])
df['actors'] = df['cast'].apply(lambda l: [] if pd.isna(l) else [i.strip() for i in l.split(",")])
df['countries'] = df['country'].apply(lambda l: [] if pd.isna(l) else [i.strip() for i in l.split(",")])
```

In [4]:

```
df.head()
```

Out[4]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	...	description	year_added	year	month	day	sea
0	s1	TV Show	3%	NaN	João Miguel, Bianca Comparato, Michel Gomes, R...	Brazil	2020-08-14	2020	TV-MA	0.0	...	In a future where the elite inhabit an island ...	2020.0	2020.0	8.0	14.0	
				Jorge	Demián Bichir,							After a					

VISUALIZING THE DATA

jupyter V Andal Priyadharshini p20038 Last Checkpoint: a day ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Markdown

In [6]:

```
display(HTML(f"""

        <ul class="list-group">
          <li class="list-group-item disabled" aria-disabled="true"><h4>Dataset preview</h4></li>
          <li class="list-group-item"><h4>Number of rows in the dataset: <span class="label label-primary">{ netdata.shape[0]:,}<
          <li class="list-group-item"> <h4>Number of columns in the dataset: <span class="label label-primary">{netdata.shape[1]]
          <li class="list-group-item"><h4>Number of unique types in the dataset: <span class="label label-success">{ netdata['typ
        </ul>

    """))
```

Dataset preview

Number of rows in the dataset: 7,787

Number of columns in the dataset: 21

Number of unique types in the dataset: 2

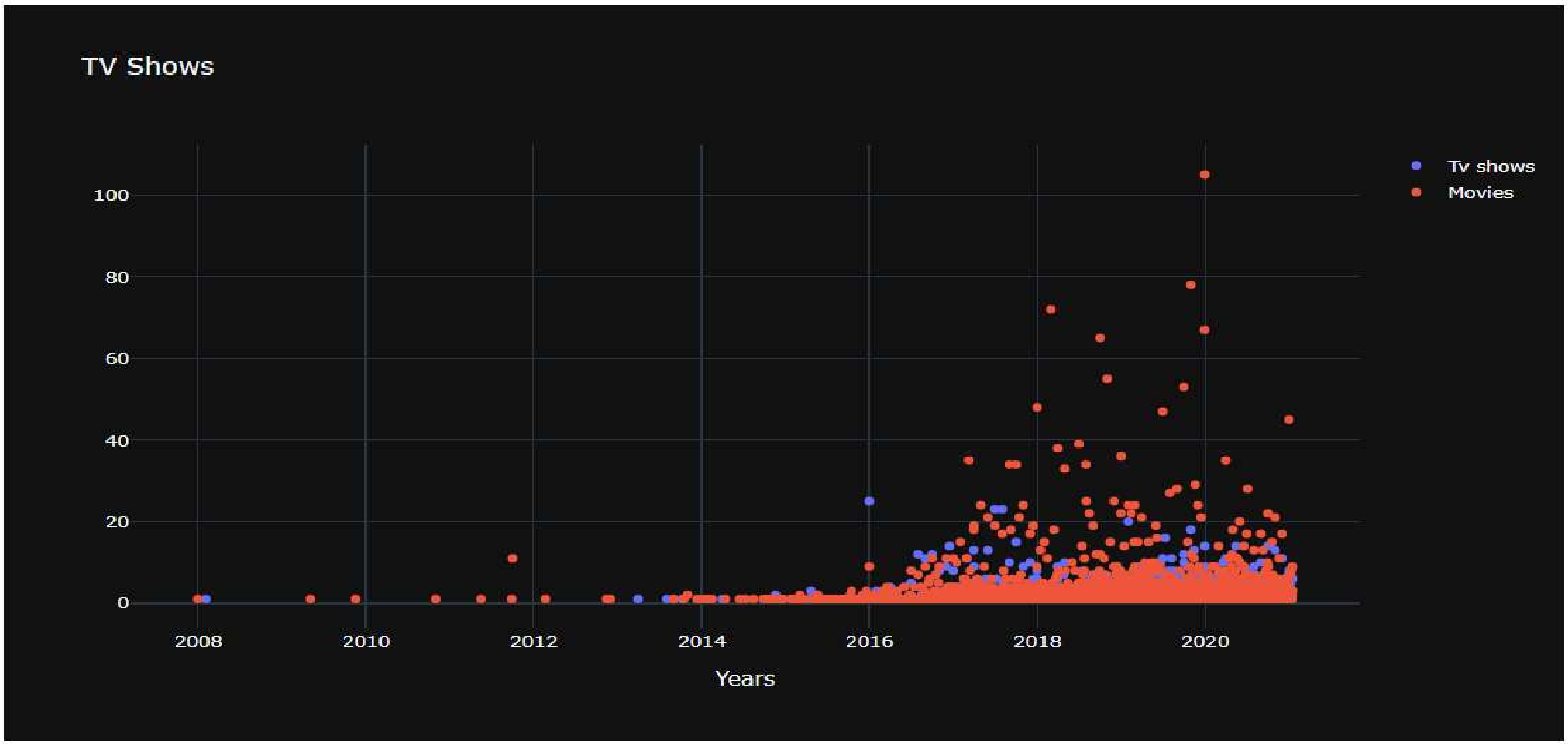
In [7]:

```
df1=df[df['type']=='TV Show']
df2=df[df['type']=='Movie']

df1=df1.groupby('date_added')['title'].nunique().sort_values()
df2=df2.groupby('date_added')['title'].nunique().sort_values()

trace1 = go.Scatter(x = df1.index,y = df1.values,mode = 'markers',name = 'Tv shows')
trace2 = go.Scatter(x = df2.index, y = df2.values, mode = 'markers', name = 'Movies')
layout = go.Layout(template= "plotly_dark", title = 'TV Shows', xaxis = dict(title = 'Years'))
fig = go.Figure(data = [trace1,trace2], layout = layout)
fig.show()
```

TV SHOWS AND MOVIES FROM DATASET



VISUALIZING THE DATA

More movies are getting released since mid 2017 than TV shows.

In [8]:
```
pd.crosstab(netdata.type,netdata.year_added,margins=True).style.background_gradient(cmap='summer_r')
```

Out[8]:

year_added	2008.0	2009.0	2010.0	2011.0	2012.0	2013.0	2014.0	2015.0	2016.0	2017.0	2018.0	2019.0	2020.0	2021.0	All
type															
Movie	1	2	1	13	3	6	19	58	258	864	1255	1497	1312	88	5377
TV Show	1	0	0	0	0	5	6	30	185	361	430	656	697	29	2400
All	2	2	1	13	3	11	25	88	443	1225	1685	2153	2009	117	7777

In [9]:
```
pd.crosstab(netdata.type,netdata.season_count,margins=True).style.background_gradient(cmap='RdYlGn')
```

Out[9]:

season_count		1	10	11	12	13	15	16	2	3	4	5	6	7	8	9	All
type																	
Movie	5377	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5377
TV Show	0	1608	6	3	2	2	2	1	382	184	87	58	30	19	18	8	2410
All	5377	1608	6	3	2	2	2	1	382	184	87	58	30	19	18	8	7787

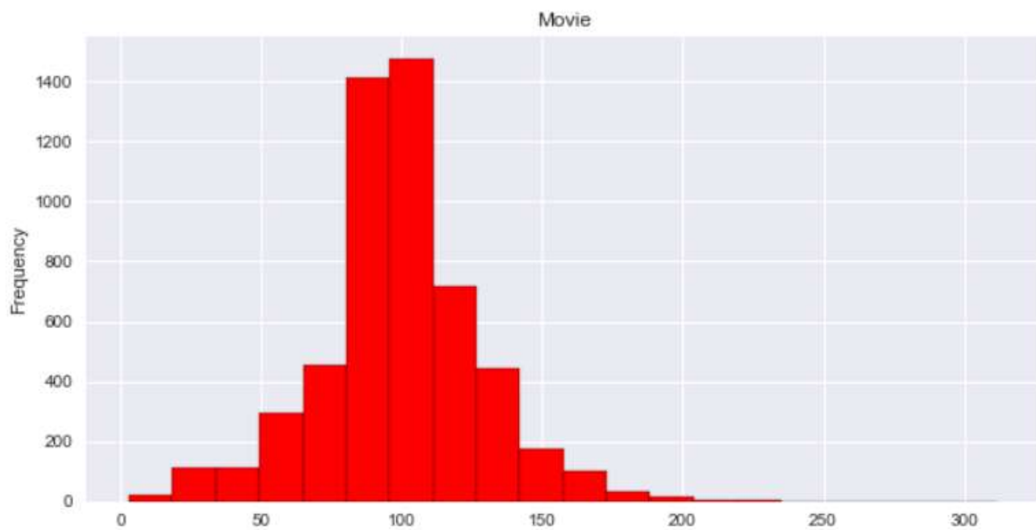
Almost all tv shows are having 1/2/3 seasons as audience binge watches the entire series in a day. Mini series will be good catch for the audiences.

MOVIE DURATION

Movie duration

In [10]:
```
f,ax=plt.subplots(1,1,figsize=(10,5))
netdata[netdata['type']=='Movie'].duration.plot.hist(ax=ax,bins=20,edgecolor='black',color='red')
ax.set_title('Movie')
```

Out[10]: `Text(0.5, 1.0, 'Movie')`



How to take in account of the description ?

First idea ...

In order to take in account the description, the movie are clustered by applying a KMeans clustering with TF-IDF weights

So two movies that belong in a group of description will share a node.

The fewer the number of films in the group, the more this link will be taken into account

**but it doesn't work because clusters are too unbalanced*

Second idea ...

In order to take in account the description, calculate the TF-IDF matrix

and for each film, take the top 5 of similar descriptions and create a node Similar_to_this. This node will be taken in account in the Adamic Adar measure.

ADAMIC ADAR MEASURE

◦

It is a measure used to compute the closeness of nodes based on their shared neighbors.

•x and y are 2 nodes (2 Movies)

•N(one_node) is a function that return the set of adjacent nodes to one_node

$$adamicAdar(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log(N(u))}$$

«say otherwise, for each node u in common to x and y, add to the measure 1/log(N(u))»

The quantity $\frac{1}{\log(N(u)}$ determine the importance of u in the measure.

•if x and y share a node u that has a lot of adjacent nodes, this node is not really relevant.

•→ N(u) is high → 1/log(N(u)) is not high

•if x and y share a node u that **not** has a lot of adjacent nodes, this node is **really** relevant.

•→ N(u) is **not** high → 1/log(N(u)) is higher

K MEANS CLUSTERING

Recomendation System

KMeans clustering with TF-IDF

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.cluster import MiniBatchKMeans

# Build the tfidf matrix with the descriptions
start_time = time.time()
text_content = df['description']
vector = TfidfVectorizer(max_df=0.4,         # drop words that occur in more than X percent of documents
                             min_df=1,       # only use words that appear at least X times
                             stop_words='english', # remove stop words
                             lowercase=True, # Convert everything to lower case
                             use_idf=True,   # Use idf
                             norm=u'l2',     # Normalization
                             smooth_idf=True # Prevents divide-by-zero errors
                            )
tfidf = vector.fit_transform(text_content)

# Clustering  Kmeans
k = 200
kmeans = MiniBatchKMeans(n_clusters = k)
kmeans.fit(tfidf)
centers = kmeans.cluster_centers_.argsort()[:,::-1]
terms = vector.get_feature_names()

# print the centers of the clusters
# for i in range(0,k):
#     word_list=[]
#     print("cluster%d:"% i)
#     for j in centers[i,:10]:
#         word_list.append(terms[j])
#     print(word_list)

request_transform = vector.transform(df['description'])
# new column cluster based on the description
df['cluster'] = kmeans.predict(request_transform)

df['cluster'].value_counts().head()
```

Out[11]:
```
6      7496
20        6
132       5
1         5
195       4
Name: cluster, dtype: int64
```

THE SYSTEM

- Import the necessary libraries.
- Import the ‘Netflix_titles.csv’ file for which we have to build the recommendation system.
- Plot the movies and tv series based on the year released.
- Plot the movies with the number of movies against the number of minutes they are running.
- Then build the actual recommendation system.
  - Explore the neighborhood of the target film → this is a list of actor, director, country, categories
  - Explore the neighborhood of each neighbor → discover the movies that share a node with the target field
  - Calculate Adamic Adar measure → final results

BUILDING THE RECOMMENDATION SYSTEM

jupyter V Andal Priyadharshini p20038 Last Checkpoint: a day ago (autosaved)

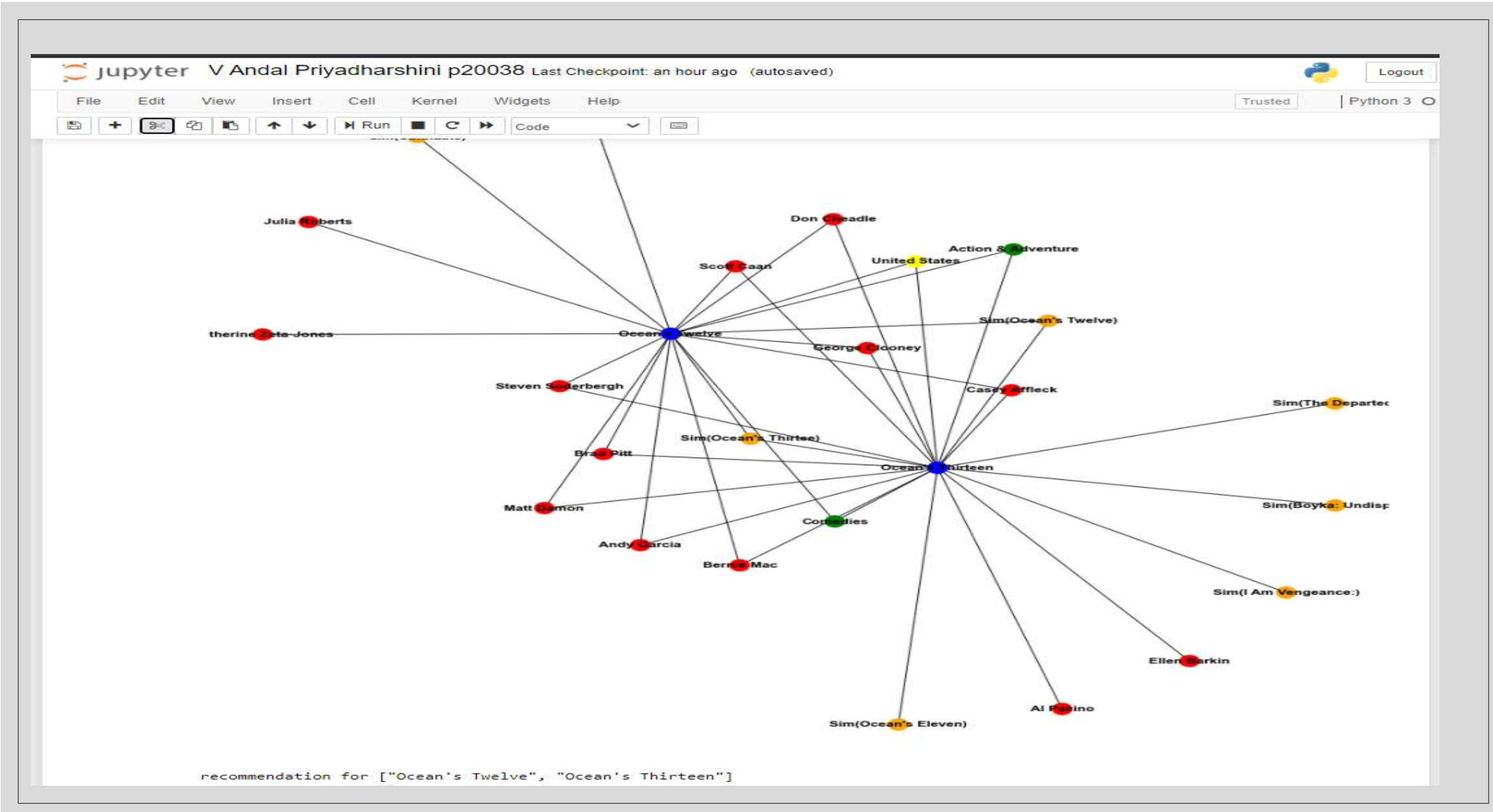
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

In [12]:
```
# Find similar : get the top_n movies with description similar to the target description
def find_similar(tfidf_matrix, index, top_n = 5):
    cosine_similarities = linear_kernel(tfidf_matrix[index:index+1], tfidf_matrix).flatten()
    related_docs_indices = [i for i in cosine_similarities.argsort()[::-1] if i != index]
    return [index for index in related_docs_indices][0:top_n]
```

In [13]:
```
G = nx.Graph(label="MOVIE")
start_time = time.time()
for i, rowi in df.iterrows():
    if (i%1000==0):
        print(" iter {} -- {} seconds --".format(i,time.time() - start_time))
    G.add_node(rowi['title'],key=rowi['show_id'],label="MOVIE",mtype=rowi['type'],rating=rowi['rating'])
#    G.add_node(rowi['cluster'],label="CLUSTER")
#    G.add_edge(rowi['title'], rowi['cluster'], label="DESCRIPTION")
    for element in rowi['actors']:
        G.add_node(element,label="PERSON")
        G.add_edge(rowi['title'], element, label="ACTED_IN")
    for element in rowi['categories']:
        G.add_node(element,label="CAT")
        G.add_edge(rowi['title'], element, label="CAT_IN")
    for element in rowi['directors']:
        G.add_node(element,label="PERSON")
        G.add_edge(rowi['title'], element, label="DIRECTED")
    for element in rowi['countries']:
        G.add_node(element,label="COU")
        G.add_edge(rowi['title'], element, label="COU_IN")

    indices = find_similar(tfidf, i, top_n = 5)
    snode="Sim("+rowi['title'][:15].strip()+")"
    G.add_node(snode,label="SIMILAR")
    G.add_edge(rowi['title'], snode, label="SIMILARITY")
    for element in indices:
        G.add_edge(snode, df['title'].loc[element], label="SIMILARITY")
print(" finish -- {} seconds --".format(time.time() - start_time))
```

```
 iter 0 -- 0.060835838317871094 seconds --
 iter 1000 -- 3.9326274394989014 seconds --
 iter 2000 -- 7.729216814041138 seconds --
 iter 3000 -- 11.593886137008667 seconds --
 iter 4000 -- 15.398740530014038 seconds --
 iter 5000 -- 19.24264883995056 seconds --
 iter 6000 -- 23.10832452774048 seconds --
 iter 7000 -- 26.91335654258728 seconds --
 finish -- 29.98518991470337 seconds --
```


jupyter V Andal Priyadharshini p20038 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Julia Roberts
Don Cheadle
Action & Adventure
United States
Scott Caan
Sim(Ocean's Twelve)
therine Zeta-Jones
Ocean's Twelve
George Clooney
Steven Soderbergh
Casey Affleck
Sim(The Departec
Sim(Ocean's Thirtee)
Brad Pitt
Ocean's Thirteen
Matt Damon
Sim(Boyka: Undisp
Comedies
Andy Garcia
Bernie Mac
Sim(I Am Vengeance:)
Ellen Barkin
Al Pacino
Sim(Ocean's Eleven)
recommendation for ["Ocean's Twelve", "Ocean's Thirteen"]

RECOMMENDATION FUNCTION

jupyter V Andal Priyadharshini p20038 Last Checkpoint: a day ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

Run Code

Recommendation Function

In [16]:
```
def get_recommendation(root):
    commons_dict = {}
    for e in G.neighbors(root):
        for e2 in G.neighbors(e):
            if e2==root:
                continue
            if G.nodes[e2]['label']=="MOVIE":
                commons = commons_dict.get(e2)
                if commons==None:
                    commons_dict.update({e2 : [e]})
                else:
                    commons.append(e)
                    commons_dict.update({e2 : commons})
    movies=[]
    weight=[]
    for key, values in commons_dict.items():
        w=0.0
        for e in values:
            w=w+1/math.log(G.degree(e))
        movies.append(key)
        weight.append(w)

    result = pd.Series(data=np.array(weight),index=movies)
    result.sort_values(inplace=True,ascending=False)
    return result;
```

Let's test it ...

In [17]:
```
result = get_recommendation("Friends")
result2 = get_recommendation("Forensic")
result3 = get_recommendation("La casa de papel")
result4 = get_recommendation("Stranger Things")
print("*"*40+"\n Recommendation for 'Friends'\n"+"*"*40)
print(result.head())
print("*"*40+"\n Recommendation for 'Forensic'\n"+"*"*40)
print(result2.head())
print("*"*40+"\n Recommendation for 'La casa de papel'\n"+"*"*40)
print(result3.head())
print("*"*40+"\n Recommendation for 'Stranger Things'\n"+"*"*40)
print(result4.head())
```

TESTING RECOMMENDATION FUNCTION

jupyter V Andal Priyadharshini p20038 Last Checkpoint: a day ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

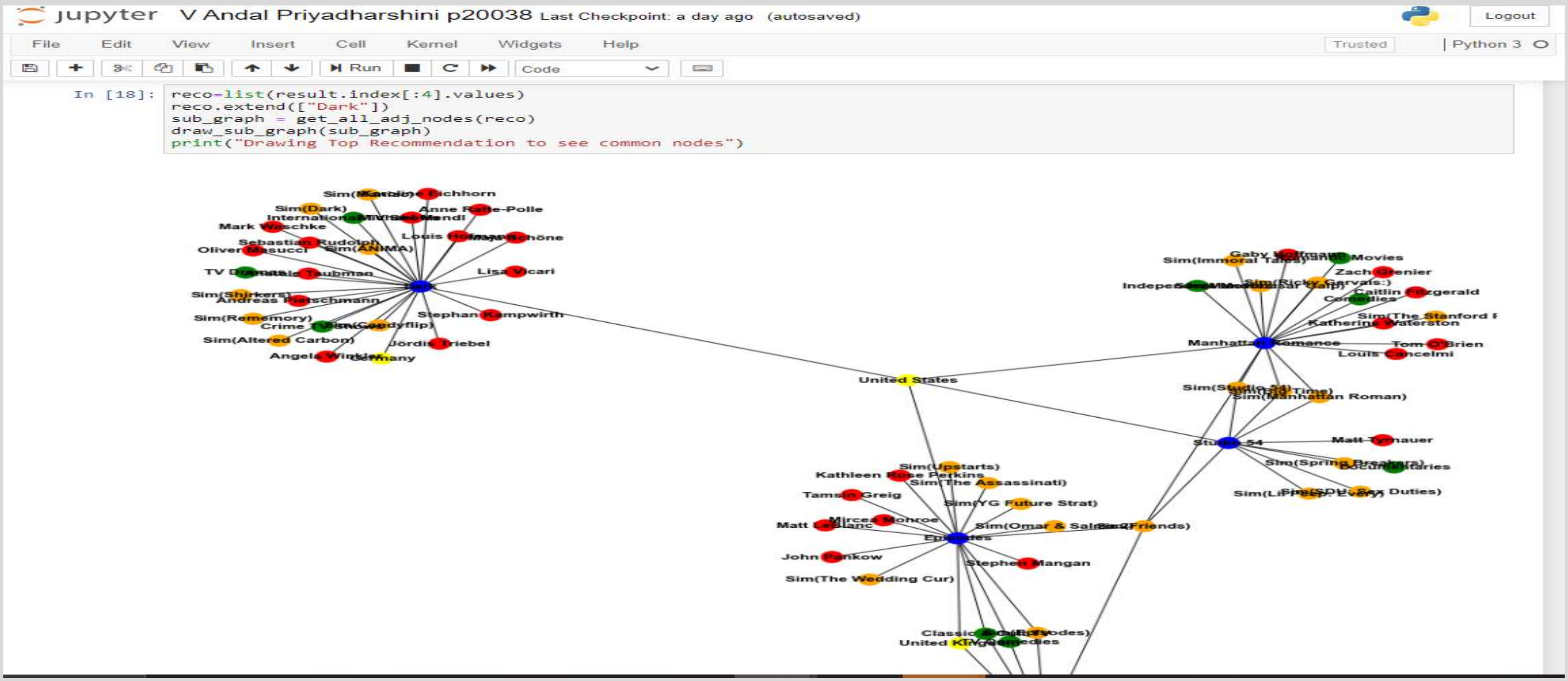
Run Code

In [17]:

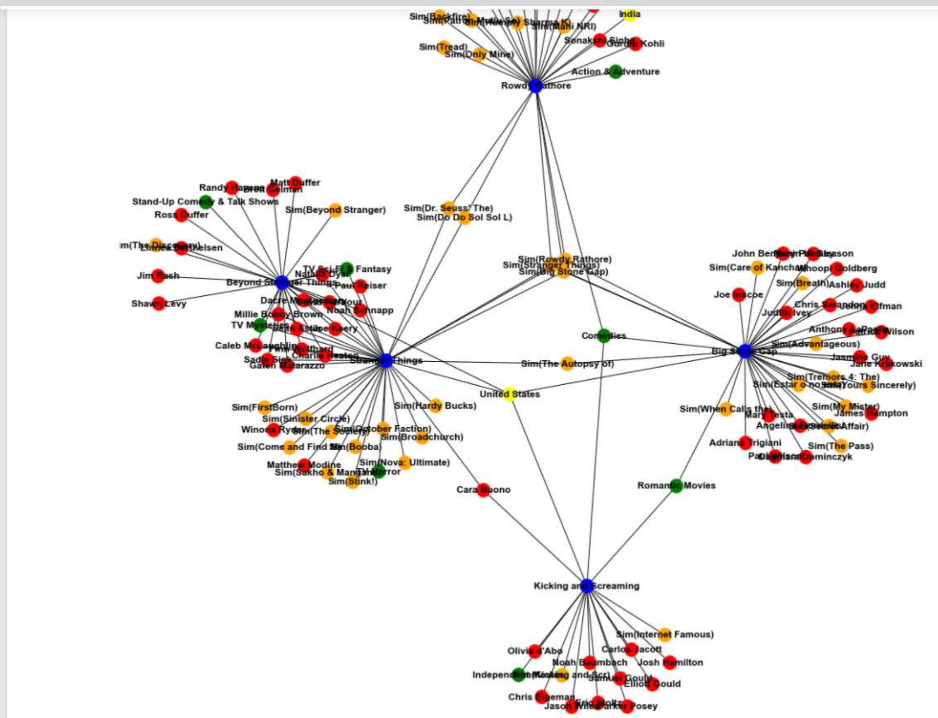
```
result = get_recommendation("Friends")
result2 = get_recommendation("Forensic")
result3 = get_recommendation("La casa de papel")
result4 = get_recommendation("Stranger Things")
print("*"*40+"\n Recommendation for 'Friends'\n"+"*"*40)
print(result.head())
print("*"*40+"\n Recommendation for 'Forensic'\n"+"*"*40)
print(result2.head())
print("*"*40+"\n Recommendation for 'La casa de papel'\n"+"*"*40)
print(result3.head())
print("*"*40+"\n Recommendation for 'Stranger Things'\n"+"*"*40)
print(result4.head())
```

```
****************************************
 Recommendation for 'Friends'
****************************************
Episodes              2.324072
Studio 54             1.797777
Manhattan Romance     1.797777
Dad's Army            1.579292
17 Again              1.566140
dtype: float64
****************************************
 Recommendation for 'Forensic'
****************************************
Uyare                  1.715900
Aadu 2                 1.715900
Unknown Origins        1.405834
Kolaiyuthir Kaalam     1.344827
Ma Chu Ka              1.344827
dtype: float64
****************************************
 Recommendation for 'La casa de papel'
****************************************
Elite              2.945666
Locked Up          2.135439
V.R. Troopers      2.076761
Coin Heist         1.674332
Mirage             1.628893
dtype: float64
****************************************
 Recommendation for 'Stranger Things'
****************************************
Beyond Stranger Things    12.047956
Rowdy Rathore              2.585399
Big Stone Gap              2.355888
Kicking and Screaming      1.566140
```

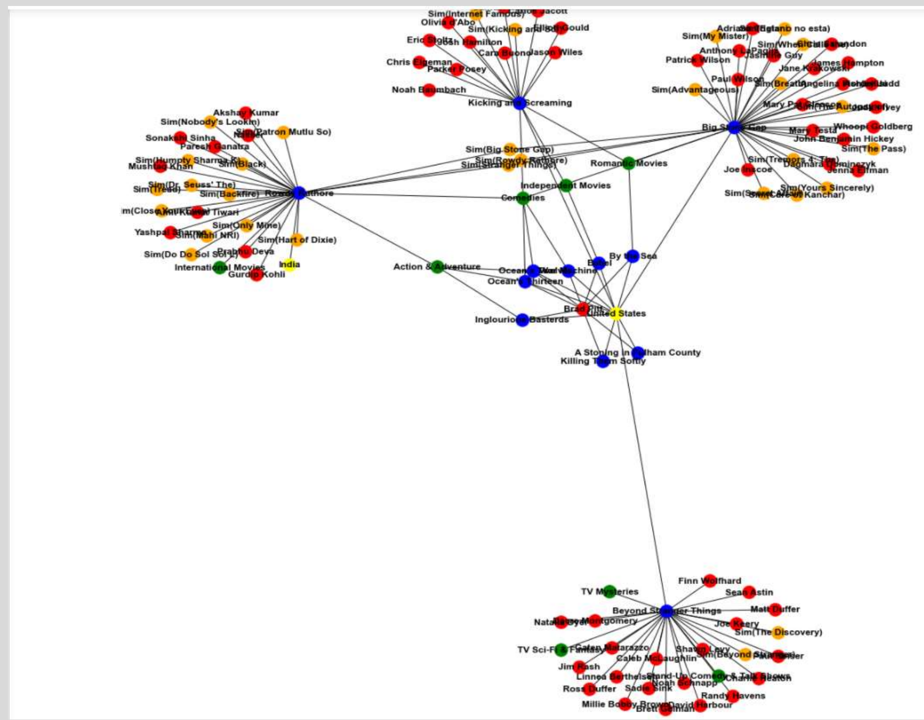
FIND COMMON NODES



Recommendation System based on title and artist



Based on series: Stranger Things



Based on actor: Brad Pitt

CONCLUSION

- We plotted a graph to see what's going on a sub-graph with only two movies
- We have created a recommendation function
- We have tested the recommendation system by calling the function.
- Drawn top recommendations, to see the common nodes by plotting it in a network.
- Thus, we have successfully built a recommendation system that recommends us movies based on our interests if we give the movie name or an actor name.

THANK YOU

By: V ANDAL PRIYADHARSHINI

SRMIST, INDIA